This Rust program is a command-line implementation of the popular game 2048. The program imports various modules and external crates that provide functionality for user input, terminal output, and game logic. The main function sets up a terminal and an event loop, and on each iteration of the loop it calls the draw method on the terminal to render the game's user interface. The draw method takes a closure as an argument, which is used to draw various elements of the user interface, such as a canvas for the game board and text for the score and current game command. The game logic is implemented in the App struct, which is created and updated in the event loop. The loop also handles user input events, such as key presses, and uses them to update the game state and trigger actions such as moving tiles or quitting the game.

The following crates are utilized:
termion: This crate provides functions and types for interacting with the terminal, such as reading user input and controlling the cursor position.
tui: This crate provides a library for creating terminal-based user interfaces, including widgets such as canvases and text blocks, and layout tools for arranging them on the screen.
std: This crate is the Rust standard library, which provides a wide range of functions and types for working with common data types, algorithms, and I/O operations.

Overall, Ive learned from this assignment a few things. Interactive terminal applications are possible utilizing TUI which is a framework to allow developers to create terminal applications that can be used similar to a GUI. I have seen examples like music players being integrated in a terminal environment which is amazing. The pain points were finding a rust program that was not utilizing deprecated crates or outdated code. It seems that many projects on Github become abandoned and it apparent by that last updates to the code being over a year old. It was a great experience modifying and breaking down the code to see how it works. I have thoroughly documented what each file does and within the code are now comments that describe the functions purpose.

Breakdown of each files:

Main.rs:

The main function of a program that uses the tui crate to create a terminal-based user interface for a game called 2048.
The main function starts by setting up the terminal for drawing, then it creates a Config object and an Events object, and an App object. The Config object specifies the tick rate, which is the frequency at which the game's logic will be updated. The Events object is used to handle user input events such as keyboard presses. The App object represents the game state and logic.
The main function then enters an infinite loop, in which it calls the draw method on the Terminal object on each iteration. The draw method takes a closure as an argument, which is used to render the game's user interface. The closure has a single argument, f, which represents the frame that the user interface will be drawn on. The closure first splits the frame into two rectangles, representing the left and right halves of the terminal window. It then

draws the game board and the game's panel using the Canvas widget from the tui crate. The game board is a grid of boxes, each representing a tile in the game. The game's panel displays the current score and other information.
The code also defines a number of functions, such as pad_str, which adds padding to a string, and score_to_color, which maps a score to a color. These functions are used to render the game board and the game's panel.

App.rs:

defines a struct called App which represents the application of the 2048 game. The App struct has several fields including x and y, which represent the x and y coordinates of the terminal window where the game will be displayed, box_size, which represents the size of each square in the game grid, game, which is an instance of the Game struct representing the game state and logic, queue, which is a vector of Commands representing the commands that will be applied to the game, and score, which represents the current score of the game.
The App struct has several methods associated with it including new, which creates a new instance of the App struct with default values, get_size, which returns the size of the game board, next, which updates the game state to the next tick based on the commands in the queue, get_score, which returns the current score of the game, is_alive, which returns a boolean indicating whether the game is still in progress or not, add_command, which adds a new command to the queue, restart, which restarts the game, get_grid, which returns the current game grid, and get_game_over_modal, which calculates the points on the game board to be displayed when the game is over.

Event.rs:

Defines an Event enum, which has two variants: Input, which represents an input event, and Tick, which represents a tick event. It also defines a Config struct which holds configuration information for the events, including the tick rate.
The Events struct is used to handle input and tick events for the game. It has several fields including rx, which is a receiver for the Event enum, input_handle, which is a handle to a thread that listens for input events, and tick_handle, which is a handle to a thread that generates tick events at a specified frequency.
The Events struct has two methods associated with it: with_config, which creates a new instance of the Events struct with a given Config object, and next, which returns the next Event in the receiver.
The with_config method creates a new channel for sending Events, and spawns two new threads: one for handling input events and one for generating tick events. The input event thread listens for keypresses from the user and sends them as Input events through the channel. The tick event thread generates Tick events at a frequency specified by the tick_rate field of the Config object, and sends them through the channel.
The next method returns the next Event in the receiver. This method can be used to listen for input and tick events in the main loop of the game.

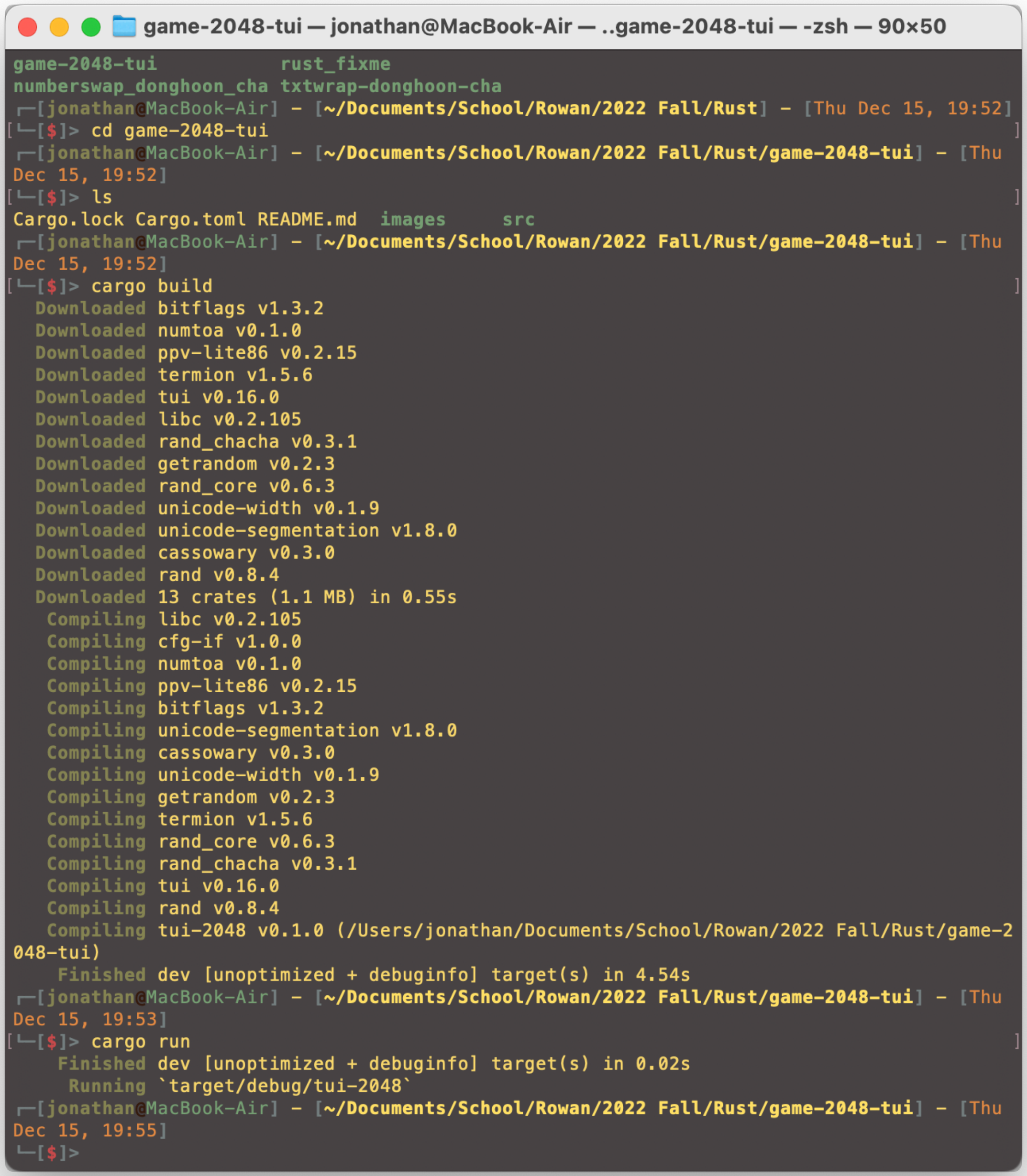
Game.rs:

The App struct represents an instance of the game 2048. It has several fields:
x and y represent the x-axis and y-axis start points for the game's user interface in the terminal.
box_size is the size of each square in the game grid.
game is an instance of the Game struct, which contains the game's state and logic.
queue is a vector of Command values that represent the player's input.
score is the player's current score.
The App struct has several methods:
new creates a new instance of the App struct with default values.
get_size returns the size of the game board, which is box_size * 4.
next updates the game state by executing the next command in the queue.
get_score returns the player's current score.
is_alive returns a boolean value indicating whether the game is still in progress.
add_command adds a new Command value to the queue.
restart resets the game to its initial state.
get_grid returns the current state of the game grid.
get_game_over_modal returns the points that make up the game over modal (a visual element displayed when the game ends).

Utils.rs:

This function is useful for comparing two slices for equality. It can be used in scenarios where you want to check if two slices contain the same elements in the same order. For example, you could use it to compare the current state of a game grid with the previous state to determine if any changes have been made.

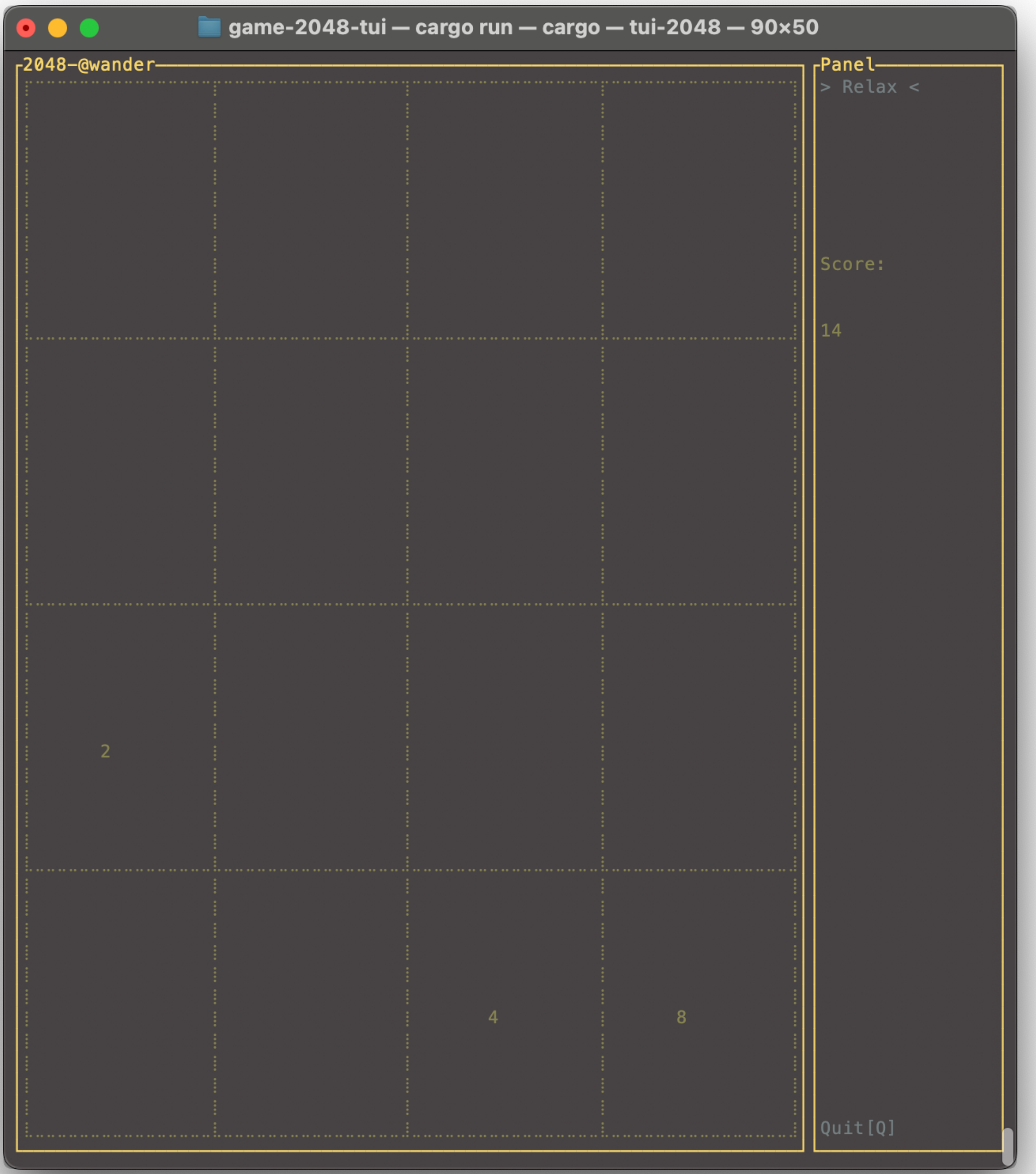
Upon building the program, the following crates were installed



game-2048-tui — jonathan@MacBook-Air — ..game-2048-tui — -zsh — 90×50

```
game-2048-tui           rust_fixme
numberswap_donghoon_cha txtwrap-donghoon-cha
┌─[jonathan@MacBook-Air] - [~/Documents/School/Rowan/2022 Fall/Rust] - [Thu Dec 15, 19:52]
[└─[$]> cd game-2048-tui                                                                 ]
┌─[jonathan@MacBook-Air] - [~/Documents/School/Rowan/2022 Fall/Rust/game-2048-tui] - [Thu
Dec 15, 19:52]
[└─[$]> ls                                                                               ]
Cargo.lock Cargo.toml README.md  images     src
┌─[jonathan@MacBook-Air] - [~/Documents/School/Rowan/2022 Fall/Rust/game-2048-tui] - [Thu
Dec 15, 19:52]
[└─[$]> cargo build                                                                      ]
  Downloaded bitflags v1.3.2
  Downloaded numtoa v0.1.0
  Downloaded ppv-lite86 v0.2.15
  Downloaded termion v1.5.6
  Downloaded tui v0.16.0
  Downloaded libc v0.2.105
  Downloaded rand_chacha v0.3.1
  Downloaded getrandom v0.2.3
  Downloaded rand_core v0.6.3
  Downloaded unicode-width v0.1.9
  Downloaded unicode-segmentation v1.8.0
  Downloaded cassowary v0.3.0
  Downloaded rand v0.8.4
  Downloaded 13 crates (1.1 MB) in 0.55s
   Compiling libc v0.2.105
   Compiling cfg-if v1.0.0
   Compiling numtoa v0.1.0
   Compiling ppv-lite86 v0.2.15
   Compiling bitflags v1.3.2
   Compiling unicode-segmentation v1.8.0
   Compiling cassowary v0.3.0
   Compiling unicode-width v0.1.9
   Compiling getrandom v0.2.3
   Compiling termion v1.5.6
   Compiling rand_core v0.6.3
   Compiling rand_chacha v0.3.1
   Compiling tui v0.16.0
   Compiling rand v0.8.4
   Compiling tui-2048 v0.1.0 (/Users/jonathan/Documents/School/Rowan/2022 Fall/Rust/game-2
048-tui)
    Finished dev [unoptimized + debuginfo] target(s) in 4.54s
┌─[jonathan@MacBook-Air] - [~/Documents/School/Rowan/2022 Fall/Rust/game-2048-tui] - [Thu
Dec 15, 19:53]
[└─[$]> cargo run                                                                        ]
    Finished dev [unoptimized + debuginfo] target(s) in 0.02s
     Running `target/debug/tui-2048`
┌─[jonathan@MacBook-Air] - [~/Documents/School/Rowan/2022 Fall/Rust/game-2048-tui] - [Thu
Dec 15, 19:55]
└─[$]>
```

Fortunately, no build errors were found and the code runs flawlessly, with the terminal accepting all user inputs.



```
2048-@wander                                      Panel
                                                  > Relax <

                                                  Score:

                                                  14

      2

                          4          8
                                                  Quit[Q]
```